FPGA-based Hardware for Physical Modelling Sound Synthesis by Finite Difference Schemes

FPGA-based Hardware for Physical Modelling Sound Synthesis by Finite Difference Schemes

Erdem Motuk, Roger Woods, and Stefan Bilbao
Sonic Arts Research Centre, Queen's University of Belfast
Belfast, Northern Ireland
E-mail: e.motuk, r.woods, s.bilbao@qub.ac.uk

Abstract

The implementation of physical models of musical instruments by finite difference methods can be highly computationally complex. This paper investigates the use of FPGAs to accelerate these numerical methods to allow real-time production of sounds for musical applications. The methodology to derive a circuit architecture that will effectively exploit the types of concurrency in the algorithm for real-time performance is described. An initial implementation on Xilinx XC2VP50 device allows computation to be performed for producing 1 second of plate sound sampled at 44.1kHz on a finite difference grid of size 100x100 in 0.84 s compared to around half an hour on a P4 Centrino 1.6 GHz laptop using MATLAB.

1. Introduction

Physical modeling is a sound synthesis technique in which the production mechanism of sound (the physical model) can be used to generate and represent a class of sounds. These physical models involve time-dependent partial differential equations (PDEs). In order to solve these equations, numerical techniques such as the finite difference (FD) methods are used. In FD methods, PDEs are approximated by recursive difference equations defined over a grid. Major drawbacks of this approach are the huge computational and memory requirements arising from high space and time sampling rates. For these reasons a real-time implementation on a single computer is not possible and other parallel implementations should be sought.

With ever-increasing capacity and addition of embedded resources such as RAMs and multipliers, FPGA devices have become a suitable platform for accelerating FD calculations. They can be used either as co-processors or a complete (System on a Chip) SoC solution to build real-time sound synthesis systems. Previous research [5] [6] [7] into FPGA-based FD hardware accelerators, have been used for calculating electromagnetic equations and the wave equation for seismic modelling. Here, the application involves audio synthesis and the FD schemes and the performance requirements involve special requirements. Furthermore, those works do not propose a design methodology for FPGA implementation to exploit different types of parallelism in the FD schemes.

In this paper, a methodology is described for the real-time FPGA implementation of a particular explicit difference scheme for a thin plate model in which the sound production mechanism can be used to approximate complex structures such as guitar bodies and percussive instruments. [3] The FD algorithm is represented at a high level and the types of concurrency explored lead to a general hardware design that satisfies the performance requirements. An initial FPGA implementation is presented along with performance figures.

2. Finite difference plate model

2.1. The plate model

The plate model to be employed is an extension of a classical Kirchoff plate with added simple linear and frequency dependent damping terms. [1]

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \nabla^4 u + c^2 \nabla^2 u - 2\sigma \frac{\partial u}{\partial t} + b_1 \frac{\partial}{\partial t} \nabla^2 u + f(x, y, t) \tag{1}$$

Here, $u(x, y, t)$ is the transverse plate deflection defined over a rectangular region $0 \le x \le L_x$, $0 \le y \le L_y$, and for $t \ge 0$. ∇^2 and ∇^4 are the Laplacian and biharmonic operators, κ^2 depends on plate stiffness and c is the propagation velocity of plane harmonic waves resulting from constant applied tension (The term with c^2 adds "membrane-like" characteristics to the stiff plate). The term involving σ, is a simple linear damping term and b_1 is the coefficient for

frequency-dependent damping. The frequency-dependent damping term allows a higher rate of loss at higher frequencies. $f(x,y,t)$ represents a time varying and spatially distributed driving force which is used for the excitation of the plate.

The PDE in (1) requires two initial conditions namely $u(x,y,0)$ and $\partial u/\partial t\,(x,y,0)$, as well as two conditions on $u(x,y,t)$ at the boundaries of the plate. This will be explained in section 2.3.

2.2. Finite difference scheme

An explicit FD scheme suitable for numerically solving equation (1) is obtained by defining a grid function $u_{i,j}^n$, which is an approximation to $u(x,y,t)$ at the space and time coordinates $x = i\Delta x$ ($0 \le i \le N_x$), $y = j\Delta y$ ($0 \le j \le N_y$), and $t = n\Delta t$, where Δt is the sampling period. The grid spacings, are taken to be equal to each other. The approximations to the time differential operators are given by:

$$\delta_t^2 u_{i,j}^n = \frac{1}{\Delta t^2}\left(u_{i,j}^{n+1} - 2u_{i,j}^n + u_{i,j}^{n-1}\right) \approx \frac{\partial^2 u}{\partial t^2} \tag{2}$$

$$\delta_{t0} u_{i,j}^n = \frac{1}{\Delta t}\left(u_{i,j}^{n+1} - u_{i,j}^{n-1}\right) \approx \frac{\partial u}{\partial t} \tag{3}$$

$$\delta_{t-} u_{i,j}^n = \frac{1}{\Delta t}\left(u_{i,j}^{n} - u_{i,j}^{n-1}\right) \approx \frac{\partial u}{\partial t} \tag{4}$$

The first two approximations are centered and second order accurate, whereas the last one is backward and first order accurate. The second order, centered approximations to the spatial difference operators are:

$$\delta_x^2 u_{i,j}^n = \frac{1}{\Delta x^2}\left(u_{i+1,j}^{n} - 2u_{i,j}^n + u_{i-1,j}^{n}\right) \approx \frac{\partial^2 u}{\partial x^2} \tag{5}$$

$$\delta_y^2 u_{i,j}^n = \frac{1}{\Delta y^2}\left(u_{i,j+1}^{n} - 2u_{i,j}^n + u_{i,j-1}^{n}\right) \approx \frac{\partial^2 u}{\partial y^2} \tag{6}$$

The Laplacian and biharmonic operator can be approximated by:

$$\delta_+^2 = \delta_x^2 + \delta_y^2 \approx \nabla^2 \tag{7}$$

$$\delta_+^2 \delta_+^2 \approx \nabla^4 \tag{8}$$

When the above operators are substituted into (1), the following explicit FD scheme is obtained:

$$\delta_t^2 u = -\kappa^2 \delta_+^2 \delta_+^2 u + c^2 \delta_+^2 u - 2\sigma\delta_{t0} u + b_1 \delta_{t-} \delta_+^2 u + f \tag{9}$$

Writing the above equation as an explicit recursion we get the formula below, where u is an internal point in the domain of the FD scheme.

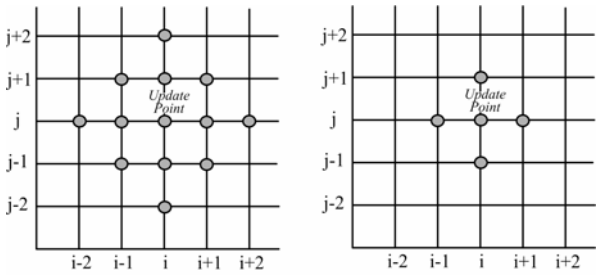
$$u_{i,j}^{n+1} = \eta \sum_{|k|+|l|\le 2} \beta_{|k|,|l|} u_{i+k,j+l}^{n} + \eta \sum_{|k|+|l|\le 1} \gamma_{|k|,|l|} u_{i+k,j+l}^{n-1} + \Delta t^2 f_{i,j}^n \tag{10}$$

where the coefficients $\beta_{0,0} = 2 - 20\mu^2 - 4(\lambda^2 + \nu)$,
$\beta_{1,0} = \beta_{0,1} = 8\mu^2 + \lambda^2 + \nu$, $\beta_{1,1} = -2\mu^2$,
$\beta_{2,0} = \beta_{0,2} = -\mu^2$, $\gamma_{0,0} = -1 + 4\nu + \sigma\Delta t$,
$\gamma_{1,0} = \gamma_{0,1} = -\nu$,
where, $\mu = \kappa\Delta t/\Delta x^2$, $\lambda = c\Delta t/\Delta x$, $\eta = 1/(1+\sigma\Delta t)$, and $\nu = b_1\Delta t/\Delta x^2$.

The stability condition for the FD scheme that can be found by spectral or von Neumann techniques is: [1]

$$\Delta x^2 \ge 2b_1\Delta t + c^2\Delta t^2 + \sqrt{(2b_1\Delta t + c^2\Delta t^2)^2 + 16\kappa^2\Delta t^2} \tag{11}$$

The recursion in (10) tells us that at each iteration step for all the points in the FD grid, the next iteration step value of a point is calculated from the previous iteration step values of the neighbouring points. Figure 1(a) and (b) shows the spatial dependencies for updating a grid point $u_{i,j}^n$ for time steps *n* and *n-1*.



(a) Time step n (b) Time step n-1

Figure 1. Grid point dependencies

2.3. Boundary conditions

As mentioned above, two boundary conditions are needed at each edge of the plate. Here the plate is simply supported at each edge, so the boundary conditions are $u(x,y,t) = \dfrac{\partial^2 u}{\partial x_n^2} = 0$ at x=0, x=L_x, y=0, and y=L_y. [2] The term $\partial^2/\partial x_n^2$ denotes the second order partial derivative with respect to the direction normal to the boundary.

For the FD scheme, the above boundary conditions translate to $u_{i,j}^n = 0$ for the points on the grid boundary. As the update of a grid point requires access to the previous values of the grid function at most two spatial steps away in x and y directions, the required values of the grid points that are not defined in the FD domain can be found from FD approximation to the second boundary condition (i.e., for the grid points adjacent to the boundary points (i=1 or i=Nx-1, and j=1 or j=Ny-1)). For example, at the bottom boundary of the grid,

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_{y=0} \approx \frac{1}{\Delta y^2}\left(u_{i,1}^n - 2u_{i,0}^n + u_{i,-1}^n\right) = 0 \qquad (12)$$

Then we can find the missing point for the update of $u_{i,1}^n$ as $u_{i,-1}^n = -u_{i,1}^n$.

2.4. Excitation and taking the output

As mentioned previously, $f(x,y,t)$ is the excitation term and is discretized as $f_{i,j}^n = f(i\Delta x, j\Delta y, n\Delta t)$ and applied as an adder term for the update of particular grid points. According to the application, this excitation can be of moving type where in each iteration period, $f_{i,j}^n$ is applied as an adder term for different grid points. Taking the output from the FD scheme corresponds to reading the value of a point or group of points in the grid in each iteration step. Depending on application, the output points may change in each iteration step.

2.5. Computational and memory requirements

From the recursive formula in (10), it can be seen that 6 multiplications and 17 additions (or 18 if an excitation is applied) are required to update a grid point. The grid size is determined by the size of the plate to be implemented and the spatial sampling step size, which is determined by the stability condition in (11). The number of points in the grid is given by $(N_x+1)\times(N_y+1) = L_x L_y / \Delta x^2$. Assuming the equality of (11) with b_1 and c taken, for simplicity, to be 0, the total number of operations per second is $23 L_x L_y / 4\kappa \Delta t^2$. For a square steel plate of side length 2m and thickness 2mm and a sampling rate of 44.1 kHz, then $15\text{x}10^9$ operations per second are needed for the FD scheme.

In addition, memory access requirement is an important parameter. Each grid point update requires reading 18 previously calculated values and writing 1 newly calculated value. This requires a large memory bandwidth when the FD grid size is large. To further complicate matters, the memory access pattern is from more than one grid point away.

3. Implementation methodology

The first step in implementing the FD scheme is to generate a high level representation and then extract the parallelism in the algorithm. After this step, design decisions can be made to derive an appropriate architecture to satisfy the real-time requirements. The final step is the implementation on FPGA where the final details of the architecture are determined according to the size of the FPGA.

3.1. Sequential algorithm

A sequential algorithm that can be used to compute the FD scheme is presented in figure 2.

```
•Do one time initialization
•For time n (time step) = 1 to nmax do
    • For i (row index) = 2 to imax do
       • For j (column index) = 2 to jmax do
            - Fetch data from memory for point update
            - Apply the update equation
            - If excitation then
               Add the excitation value to the result
            - Write back the result to memory
       End
    End
End
```

Figure 2. Sequential algorithm

3.2. Parallelization

As the FD scheme is an explicit finite difference scheme, temporal independence exists in the computation in an iteration period. This means that a grid point update requires only the previously calculated values and the order that the grid points are updated at a particular iteration step is not important. These properties allow parallelization at different levels. In order to see the parallelism more clearly, the sequential algorithm can be represented by a data flow network. Figure 3 shows the data flow representation of the computation for the update of a grid point and shows that the arithmetic operations can be computed in parallel and in a pipelined manner. This constitutes the first type of concurrency that can be applied for the implementation. The representation also provides a better view of the design space, which will be necessary to make design decisions such as the number of adders and multipliers operating in parallel at design space exploration step.

The second type of concurrency in the algorithm is the data parallelism, which comes from the fact that the same operations are applied to every point in the grid for the update in every iteration period. Therefore, as each node is performing the same operation, the nodes can be computed in parallel. However, as the hardware will be operating at substantially greater throughput rate than the sampling rate (44.1 kHz) and the FPGA hardware will be limited in size, then one processing FPGA will most likely be used to compute many grid points. This strategy is also known as domain decomposition, where the domain of the FD scheme is partitioned into blocks which are assigned to individual processing elements (PEs) connected by an interconnect network. [4] The partitioning can be done in different ways depending on the shape of the domain. For a regular rectangular domain,

partitioning can be in 1-D or 2-D. An example of domain decomposition is shown in Figure 4 where 2-D partitioning is applied to an FD grid.

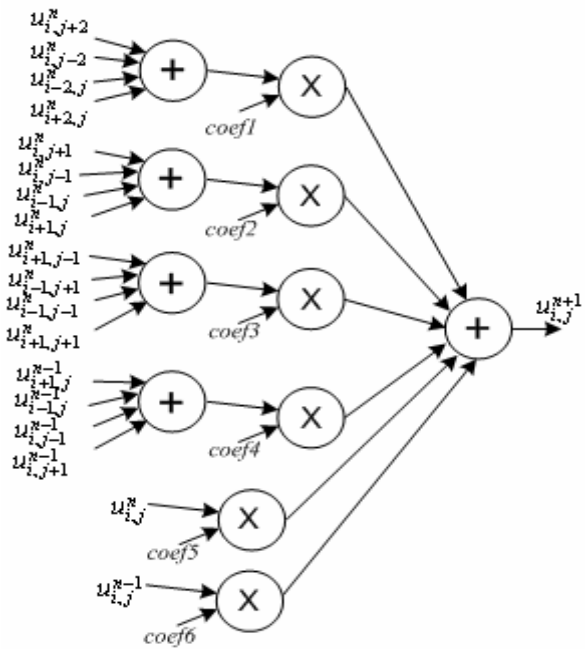


Figure 3. Dataflow graph for the update of a grid point

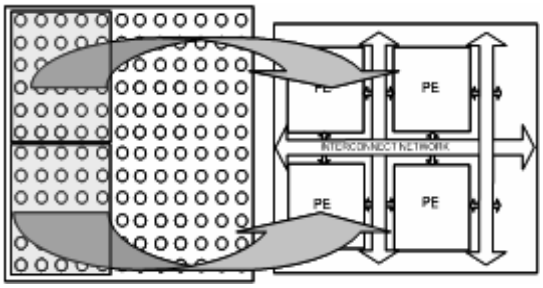


Figure 4. 2D partitioning/mapping of a FD grid

The extent to which the two mentioned types of concurrency are exploited depends on several factors related to the FD scheme and the FPGA device. These will be further explained in the next section.

4. Hardware design

Initial hardware decisions can be made by considering the specifications of the FD algorithm, and the ways the concurrencies in the algorithm can be exploited. The first parameter is the grid size which is determined by the size of the plate and the spatial sampling rate. Equation (11) gives the limit of the spatial step size in terms of the time sampling period and the material and damping parameters to achieve stability. For example, a square brass plate of size 2mx2m and thickness of 1mm will have approximately 10,000 grid points when the time sampling rate is 44.1 kHz and the parameters b_1 and c taken as zero for simplicity.

The second concern is the time sampling rate, Δt, which determines the duration of an iteration step and for real time implementation, also determines output rate. Therefore, the time it takes to update all the points in the grid has to be smaller than the time sampling period. For audio applications time sampling rate can be taken as the CD audio sampling rate of 44.1 kHz.

When the size of the FD grid is large, in order to satisfy the performance requirement, the two types of parallelism in the algorithm that are mentioned previously should be both exploited.

4.1. General architecture

To exploit the data parallelism, the design should include PEs connected by an interconnection network. This kind of architecture is shown in Figure 5 for 1-D and 2-D network topologies. The PEs are responsible for updating the grid points. The main controller generates a start signal for the PEs at the beginning of every iteration step to synchronize the operation of the PEs. It also communicates with the host computer to receive the excitation and output the results. For the initialization of the hardware, the main controller receives the parameters form the host computer and sends these to the PEs.

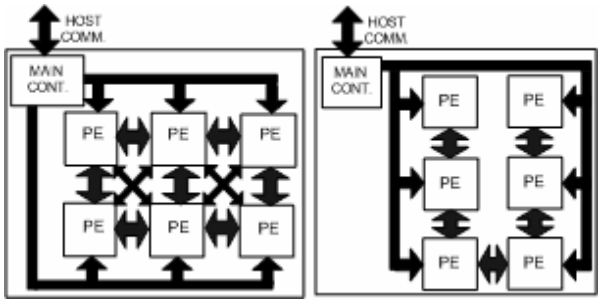


Figure 5. Interconnection architecture for 1-D and 2-D network topologies

4.2. Processing elements

The general architecture of a PE (Figure 6) consists of a memory unit to store the grid point values, an operational unit to apply the update operations, and a controller to schedule operations and control communication with the main controller and neighbouring PEs. The size of a PE depends on the extent to which the parallelism in the update operations is exploited (determining numbers of adders and multipliers) and the number of points that is assigned to the PE as the point values will be stored in memory.

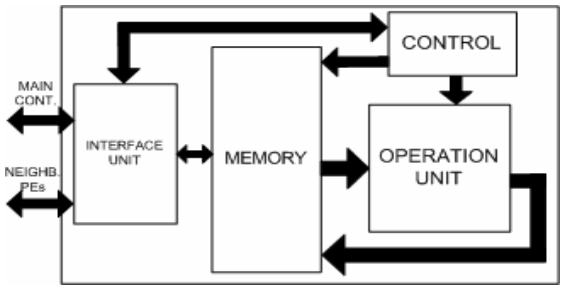


Figure 6. General architecture of a PE.

4.3. Memory configuration

As it is second order in time dimension, the FD scheme involves calculating the next iteration step values, (u^{n+1}) by using the previous (u^{n}) and two-previous iteration step (u^{n-1}) values. Therefore, there exist three sets of data corresponding to the grid. As can be seen from figure, the FD scheme can be separated into two kernels that operate on the data sets u^{n} and u^{n-1} independently. In order to exploit this parallelism each data set can be stored in separate memory blocks.

In Figure 7, the temporal nature of the memory access in the FD scheme is shown when each data set is stored in separate memory blocks. In order to avoid memory transfers between memory blocks, the kernel of the FD scheme can access the memory blocks interchangeably in each iteration step as shown. The order in which the grid points are stored in the memory is another concern. A 2-D array of grid point values will be stored in a one dimensional memory array. The most obvious choice is to store the values in a row-wise fashion (as shown in Figure 8 for a grid of size 6x6). The number pairs above the circles show the locations of the points in the grid and the numbers inside them show their locations in the memory. The memory accesses patterns in the grids storing values corresponding to two time steps for the update of a point stored in memory location 14 are also shown.

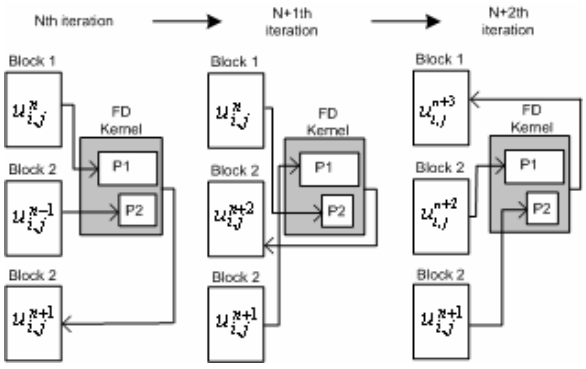


Figure 7. Temporal memory access pattern

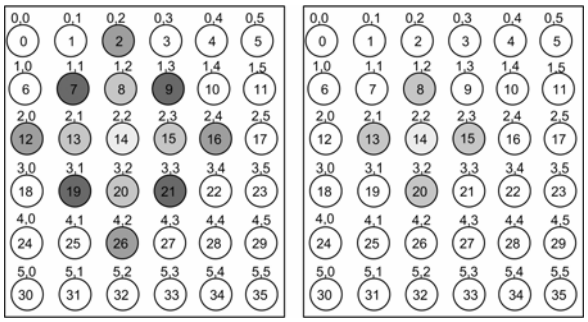


Figure 8. Memory storage pattern

4.4. Communication

Being 4th order in space, the FD scheme involves the previous values of the grid function at most two spatial steps away in x and y directions. Thus, when the FD domain is partitioned into sub-grids and assigned to PEs, each PE has to have access to the values of the boundary points (ghost points) that are originally assigned to the neighbouring sub-grids. Figure 9 shows ghost points for the two time steps, corresponding to a sub-domain which is obtained from 2-D partitioning.

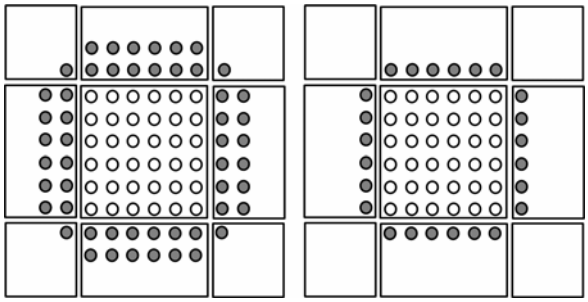


Figure 9. Ghost point conditions

In each iteration period the values of the ghost points have to be transferred between the neighbouring PEs. The number of these points depends on the size of the sub-domain and the type of partitioning. To keep the communication local, neighbouring sub-domains are mapped on to neighbouring PEs in the hardware design. The global communication in the design is between the main controller and the PEs, which involves taking the output, the excitation, and the initialization.

5. FPGA implementation

In this section, the general architecture in Section 4 is further detailed for FPGA implementation taking into account logic area, on-chip memory size and structure, on-chip multipliers, and the pin count.

5.1. Memory system

In order to reduce the impact of memory bandwidth bottleneck on the performance of the FPGA implementation, several strategies are employed. The first involves using the on-chip memory as it is much faster than external memory and can be used to store all or part of grid point values. For example, the Xilinx XCV2P50 FPGA has a total of 4176 kBits of dual-port on-chip Block RAM organized in 232 18kbit blocks. [8] Therefore, when a grid point value is represented by 18 bits and on-chip memory is allocated for storing values for u^{n-1}, u^n and u^{n+1}, up to approximately 78000 grid points can be stored on chip. This will even allow us to build multiple plates on a single chip.

The second strategy is to further increase the memory bandwidth by adding a faster and smaller memory to the memory hierarchy as shown in figure 10. This fast memory block will be between the Block RAMs and the operation units, and can be implemented with the LUTs on the FPGA device. One reason for introducing this memory block is to provide the operational unit with all the values needed for the update at each clock cycle. Dual-port Block RAM can allow only two memory reads in one clock cycle. Another reason is to make the memory access in a unit stride, thus simplifying address generation logic, which can incur performance penalties.

The architecture of the memory block is shown in Figure 10 and consists of two sets of register banks that act as two moving windows over two grids and the shift registers that enable this movement. The upper window moves over the set of u^n, while the lower one moves over the set u^{n-1}. The register banks output the exact pattern of values required for updating a grid point. In this way, the operational unit can have access to all of the 18 values needed for a point update at every clock cycle.

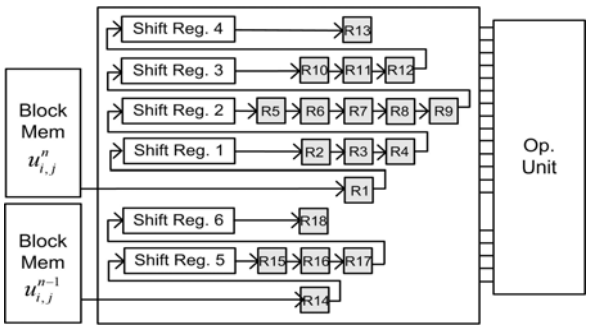


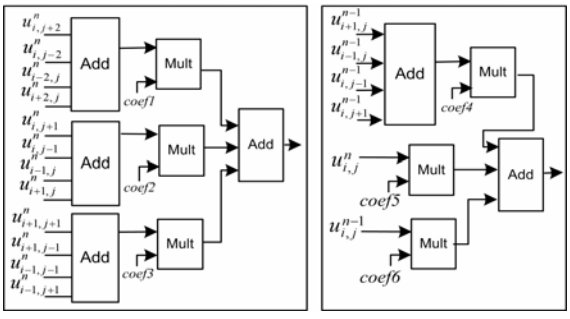
Figure 10. Memory architecture

At each clock cycle, values of the points stored in the memories will be loaded consecutively to the registers R1 and R14, and the values stored in all registers will be shifted to the next register or to the shift registers. Only one port of the dual-port Block RAM is used to perform this operation. The length of the shift registers are determined by the size of the grid. For a grid of size N_xxN_y, shift registers 1, 4, 5, and 6 will be of length N_x-2, and shift registers 2 and 3 will be of length N_x-4.

As mentioned in Section 4.3, the outputs from the Block RAMs that hold the grid point values corresponding to the three different time steps are used interchangeably for the two kernels. For this reason, two multiplexers (MUXes) have to be placed between the Block RAMs and the intermediate memory block. The same applies to the input to the Block RAMs from the operational unit.

5.2. Design of a PE

The remaining parts of a PE are the operational unit and the controller that schedules the computation and the communication with the main controller and the neighboring PEs. When the parallelism of Figure 3 is exploited fully and all operations are pipelined, an update of a grid point takes only one clock cycle. However, this requires 6 multipliers and 17 of 2-input adders to be implemented on each PE. As the logic size and the number of multipliers on a FPGA are limited, this will not be very feasible especially when the operational units are implemented as floating point units. With the graph transformations, scheduling and hardware sharing the number of adders and multipliers can be reduced to some extent. For example, only 4 multipliers can be implemented to produce a result every clock cycle. However, this might incur extra penalties in terms of complex controller and routing, which reduces clock frequency post synthesis.



(a) Top part (b) Bottom part

Figure 11. PE computation

The strategy here is to keep the control logic simple and reduce the number of adders and multipliers by compromising on the throughput. In order to do this we first cut the graph in Figure 2 horizontally into top and bottom parts which are shown in Figure 11(a) and (b). From the figures we can see that for each part one 4-input adder, one

multiplier and one accumulator is needed to produce a result in every three clock cycles. We can recoup this performance loss with the increased number of PEs that can fit on a single device.

The final design for the operational unit is shown in Figure 12. The multiplexers in the design and the resetting of the accumulators are controlled by the controller unit. In addition to this, every operation unit begins executing after receiving a start signal from the controller and after each execution sends a signal to the controller to notify that the results are ready. The controller then sends a start signal to the next operation unit on the dataflow path. The addresses for reading from and writing to the memories are also generated by the controller.

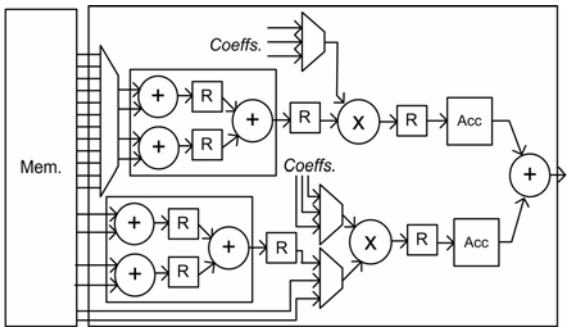


Figure 12. Final design of PE

The communication between the PEs is point-to-point and therefore, a PE should have separate input and output channels for each of its neighbours. For the inter-PE communication, the memory of a PE should be augmented by the ghost points. In order to do this another memory block is allocated for the ghost points. The input to the intermediate memory block is multiplexed to select between the inner points and ghost points during computation. At the start of each iteration period, ghost points are transferred between the PEs, and the computation starts after this. In order to implement the transfer a simple handshaking protocol made up of receive and transmit signals is used. This type of communication incurs an extra overhead for transfer of ghost points at each iteration step. The overhead depends on the number of ghost points transferred which in turn depends on the size of the sub-domain that is mapped on to a PE, and the topology of the network.

5.3. Design of the main controller

As mentioned previously, the communication with the host computer is only done through the main controller. The communication with the host computer is asynchronous which depends on handshaking through receive and transmit signals.

The initialization phase consists of the main controller getting parameters from the host computer and then sending these to the PEs. The parameters sent to the PEs are coefficients of the FD scheme, and the size of the sub-domain that is assigned to a particular PE, which is sent as number of points in x and y coordinates. The main controller employs a state machine to send the parameters on at a time to the PEs through a dedicated parameter sending bus. After all the parameters are sent, the main controller issues a start signal to all the PEs.

6. Implementation Results

The design units were coded in VHDL, simulated using ModelSim 5.8a and synthesized for a Xilinx XC2VP50 FPGA using SynplifyPro 7.6.1. Dual-port Block RAMs are generated with the Core Generator tool from Xilinx. For the multipliers, the MULT18x18S primitive is used with 18-bit fixed point representation. For each PE, 4 Block RAM parts, and 2 embedded multipliers are used. The shift registers in the intermediate memory block are implemented to be addressable using the SRLC16E primitives, which use 4-input LUTs on the FPGA device. In order for the hardware to be able to compute FD domains of different sizes without re-synthesis, the lengths of the shift registers are fixed as 32 in the implementation. This allows mapping of sub-domains having a maximum of 30 points in the horizontal direction. With 3 Block RAM parts assigned to a PE for holding the inner point values and 1 part for the ghost points, the maximum number of grid points that can be mapped on a PE is 1024.

The total number of clock cycles to compute an iteration step can be found by the formula, $N_{total} = N_{comp.} + N_{comm.} = n_s x3 + l_{comp} + n_g + l_{comm..}$, where n_s is the number of points in the sub-domain, n_g is the number of ghost points, and $l_{comm.}$ and $l_{com.}$ are the communication and the computation latencies respectively. $l_{comm.} = N_{neigh.} x3$, where $N_{neigh.}$ is the number of neighbouring PEs in the network. $l_{comp.} = l_{pipe.} + l_{sreg.}$, where $l_{pipe.}$ is the pipeline latency of the operation unit which is equal to 7, and $l_{sreg.}$ is the latency caused by loading the shift registers before starting calculation, which is equal to 3x ($n_{horiz.}$+4) if the sub-domain doesn't have neighbours above, or 4x ($n_{horiz.}$+4) if it has. Table 1 presents the synthesis results for a PE on XC2VP50.

Table 1. Synthesis results for a PE

Slices	LUTS		Clock (MHz)	% Slices
	Logic	SRL16E		
1153	1226	252	204.9	4

In order to implement a 100x100 square grid, a network of 10 PEs connected as a 1-D array can be used. For this network topology, the FD grid can be partitioned into 10 columns, each having 1000 points. Every sub-domain will then have 10 points in the horizontal direction and 100 points in the

vertical. The number of ghost points to be transferred will be at most 400. Table 2 shows the $N_{comp.}$, $N_{comm.}$, and N_{total} values for the network. The attainable output rate (f_{out}), which is the clock frequency divided by N_{total}, and the time it takes to produce 1s of sound sampled at 44.1kHz (t_{1s}) are also shown when the network is clocked at 180 MHz.

Table 2. Performance results for the network

$N_{comp.}$	$N_{comm.}$	N_{total}	$f_{out.}$ (kHz)	t_{1s} (s)
3037	406	3443	52.28	0.84

From the results it can be seen that the hardware implementation can easily reach the performance of 12.1×10^9 OPS without using full logic and memory resources on XC2VP50. The performance can further be increased by making FPGA implementation level design decisions such as increasing the number of multipliers and adders. For example the throughput for updating a grid point will be 2 clock cycles if 1 extra multiplier, 1 accumulator and 2 adders are used. However, the control logic should be carefully designed in order not to decrease the attainable clock frequency. The methodology proposed in this paper makes it easier to make design decisions at every level for improving the performance.

The performance improvement over sequential computer implementations becomes more apparent when the results are compared against the implementation on P4 Centrino 1.6 GHz laptop with 512 Mbyte RAM using MATLAB, where it takes 2108 seconds to produce 1s of sound. However, it should be noted that MATLAB uses floating point numbers and arithmetic. The floating point hardware implementation can be done by replacing the fixed point operational units with floating point cores. We are currently working on this implementation.

7. Conclusion

In this paper we presented the design steps taken for accelerating the implementation of FD schemes for plate sound synthesis by parallelization on an FPGA platform. The results show that real-time performance for a big FD grid can easily be achieved.

The main emphasis in this paper is on the implementation methodology and how to exploit different types of concurrency in the algorithm to get the required performance with the constraints of the hardware platform. The design space for the FPGA implementation is huge; therefore we first elaborated on the parallelism in the algorithm to make high level design decisions. Then, these high level design decisions are detailed during the FPGA implementation.

8. References

[1] S. Bilbao, "A Finite Difference Plate Model", to appear in *ICMC 2005*, Barcelona, Spain.

[2] K. Graff., *Wave Motion in Elastic Solids*, Dover, New York, USA, 1975.

[3] A. Chaigne and C. Lambourg, "Time Domain Simulation of Damped Impacted Plates. I. Theory and Experiments", *J. of Acoust. Soc. Am.*, vol. 109(4), pp. 1422-32, Apr. 2001.

[4] E. Acklam and H. P. Langtangen, *Parallelization of Explicit Finite Difference Schemes via Domain Decomposition*, Oslo Scientific Computing Archive, 1999.

[5] J. P. Durbano, *et al.*, "Hardware Implementation of a Three-dimensional Finite-difference Time-domain Algorithm", *IEEE Ant. and Wireless Propagation Letters*, Vol. 2, No.1, pp. 54-57, 2003

[6] W. Chen, *et al*., "An FPGA Implementation of the Two-dimensional Finite-difference Time Domain (FDTD) Algorithm", in Proc. FPGA, pp. 213-222, Monterey, USA, 2004.

[7] C. He, *et al.*, "Accelerating Seismic Migration Using FPGA-based Coprocessor Platform", in Proc. FCCM, pp. 207-216, Napa, USA, 2004.

[8] Xilinx, Virtex II Pro FPGA User Guide, www.xilinx.com